

IN THE SPECIFICATION:

Page 4, lines 16-25: Please replace the paragraph with the following:

The following presents a simplified summary of the invention in order to provide a basic understanding of some aspects of the invention. This summary is not an extensive overview of the invention. It is intended neither to identify key or critical elements of the invention nor to delineate the scope of the invention. Rather, the primary purpose of this summary is to present some concepts of the invention in a simplified form as a prelude to the more detailed description that is presented later. The invention relates to network controllers and other peripheral devices, as well as methods for transferring incoming (e.g., receive) data status information entries or other data from the peripheral to a host ~~system~~ memory, in which partial cache line writes are reduced, thereby improving system efficiency and system bus bandwidth.

Page 4, lines 26 – page 5, line 3: Please replace the paragraph with the following:

In one aspect of the invention, a method is provided for transferring data entries from a peripheral to a data queue in a host ~~system~~ memory. The method comprises determining a lower limit on a number of available data entry positions calculated by adding the number of unused data entry positions in the data queue, and selectively transferring a current data entry to the host ~~system~~ memory using a full cache line write if the lower limit is greater than or equal to a first value, the first value taking on the value of the number of unused status entry positions remaining in the cache line. The method may be employed in transferring any type of data from the peripheral to the host memory, for example, incoming (e.g., receive) data status entries including information related to the status of one or more data frames received by the peripheral from a network.

Page 5, lines 4- 14: Please replace the paragraph with the following:

In one implementation illustrated and described below, the data entries are incoming data status entries having lengths smaller than a cache line, where the data queue is an incoming data status ring in the host ~~system~~ memory. In this example, a lower limit on a number of available incoming data status entry positions in the incoming data status ring is determined, and a current incoming data status entry is selectively transferred to the host ~~system~~ memory using a full cache line write if the lower limit is greater than or equal to the first value. Where the lower limit is less than the first value, the incoming data status entry is transferred to the host ~~system~~ memory using a partial cache line write. The lower limit determination thus allows the use of full cache line writes to improve system efficiency and throughput without the risk of overwriting receive status entries that have not been processed by the host CPU.

Page 6, lines 3- 16: Please replace the paragraph with the following:

Where the lower limit is greater than or equal to the first value, the method thus provides full cache line writes to transfer a current incoming data status entry or other data entry to the host memory queue. For transferring receive status entries into a receive status ring queue, the peripheral may not know whether the host has processed the previous entries in the status ring cache line. Accordingly, one implementation of the method provides for selectively transferring the current incoming data status entry and any previous incoming data status entries for the current cache line to the host ~~system~~ memory using a full cache line write. The remaining portions of the cache line may be padded with zeros or any other data, since the host will not be expecting valid data in the data entry positions following the current entry. Accordingly, the peripheral may maintain a current cache line copy including all the previously written status entries for that cache line. Thereafter, if another status entry is to be written to the same cache line of the status ring, it is added to a current cache line copy, and the entire current cache line copy is written to the host memory using a full cache line write.

Page 6, lines 17-30: Please replace the paragraph with the following:

In another aspect of the invention, a peripheral system is provided, such as a network controller or other peripheral, comprising a descriptor management system. The descriptor management system is adapted to determine a lower limit on a number of available data entry positions in a data queue in a host ~~system~~ memory, and to selectively transfer a current data entry to the host ~~system~~ memory using a full cache line write if the lower limit is greater than or equal to a first value. In a network controller illustrated and described below, the data entries are incoming data status entries and the data queue is an incoming data status ring in the host ~~system~~ memory. In this implementation, the descriptor management system determines a lower limit on a number of available incoming data status entry positions in the incoming data status ring, and selectively transfers a current incoming data status entry to the host ~~system~~ memory using a full cache line write if the lower limit is greater than or equal to the first value. If the lower limit is less than the first value, the descriptor management system uses a partial line write to transfer the current incoming data status entry to the host memory.

Page 7, lines 23-25: Please replace the paragraph with the following:

Fig. 1F is a flow diagram illustrating an exemplary method of writing incoming data status entries from the network peripheral to the host ~~system~~ memory in accordance with the present invention;

Page 8, lines 3-5: Please replace the paragraph with the following:

Fig. 5A is a schematic diagram illustrating a control status block in the host ~~system~~ memory with pointers to descriptor rings and receives status rings in the host system of Fig. 2;

Page 8, lines 8-9: Please replace the paragraph with the following:

Fig. 5C is a schematic diagram illustrating descriptor management system unit registers in the network interface system of Fig. 2;

Page 8, lines 10-12: Please replace the paragraph with the following:

Fig. 5D is a schematic diagram illustrating an exemplary transmit descriptor ring in the host system memory and pointer registers in the descriptor management system unit of the network interface system of Fig. 2;

Page 8, lines 19-21: Please replace the paragraph with the following:

Fig. 5H is a schematic diagram illustrating an exemplary receive descriptor ring and receive status ring in the host system memory, as well as pointer registers in the descriptor management system unit of the network interface system of Fig. 2;

Page 8, lines 22-24: Please replace the paragraph with the following:

Fig. 5I is a schematic diagram illustrating an exemplary receive status ring in host system memory and pointer registers in the descriptor management system unit in the network interface system of Fig. 2;

Page 8, lines 25-26: Please replace the paragraph with the following:

Fig. 5J is a schematic diagram illustrating an exemplary receive status ring entry in the host system memory;

Page 8, lines 27-29: Please replace the paragraph with the following:

Fig. 6 is a schematic diagram illustrating an exemplary mapping of receive status entries in a receive status ring in host ~~system~~ memory with cache lines in the host system cache memory; and

Page 9, lines 1-3: Please replace the paragraph with the following:

Figs. 7A-7D illustrate the contents of an exemplary current cache line copy in the descriptor management system unit of the present invention as receive status entries are written to the receive status ring in the host ~~system~~ memory.

Page 9, lines 9-20: Please replace the paragraph with the following:

Referring initially to Figs. 1A and 1B, a host system 1 is illustrated having peripherals 2 and 3 in which one or more aspects of the present invention may be carried out. The host system 1 comprises a ~~PCI-X~~ system bus 6 to which various components in the system are coupled, including a network peripheral 2 or other peripheral 3, a shared host main memory 4, a host bridge 10, and a memory/cache controller 5. The host bridge 10 interfaces the bus 6 with a host processor 7, which is also coupled with the ~~shared host~~ memory 4 via the memory/cache controller 5 (Fig. 1B). The network peripheral 2 interfaces the host processor with an external LAN or other type network 8, and one or more other peripherals 3 interface the host processor 7 with external devices (not shown). The host system 1 also comprises a cache memory 9, which may be operated as a write-back cache or a write-through cache for improving processor efficiency, wherein the memory controller 5 may also operate as a cache controller.

Page 9, line 21 – page 10, line 2: Please replace the paragraph with the following:

Several aspects of the invention are hereinafter illustrated and described in the context of the exemplary network interface peripheral 2, which may be implemented as a single chip network controller with various interface devices, connectors, power circuitry, etc. on a network interface card (NIC) coupled to the ~~PCI-X~~ system bus 6 in the host system 1. However, the features of the invention are generally applicable to other types of peripherals, wherein the descriptor management aspects illustrated and described herein may be employed for exchanging data between any peripheral and a host such as the host processor 7. Further, the features of the invention may be employed in association with any host system in which any bus architecture, memory systems, and memory/cache controllers are used, wherein all such applications and implementations are contemplated as falling within the scope of the invention and the appended claims.

Page 10, lines 3-14: Please replace the paragraph with the following:

As illustrated in Fig. 1B, the host processor comprises peripheral driver software 11, which may communicate with the host ~~shared~~ memory 4 using the memory/cache controller 5 and the network peripheral 2 *via* the host bridge 10. The driver 11 provides interface functionality to interface the host processor and other software running therein (*e.g.*, upper layers in a network stack) with the network interface peripheral 2, which in turn, interfaces the driver 11 with an external network (*e.g.*, network 8 of Fig. 1A). While the host processor 7 and the driver 11 may quickly communicate with the host ~~shared~~ memory 4 *via* the memory controller 5 and/or the host bridge ~~local bus~~ 10, communication with the peripheral 2 *via* the system bus 6 is less efficient. Such communication is performed using I/O read and write operations, wherein I/O read operations are particularly costly in terms of processor efficiency since the host processor 7 must wait for a response by the peripheral 2.

Page 10, lines 15-24: Please replace the paragraph with the following:

In accordance with the present invention, a descriptor system is provided for transferring incoming data from the peripheral 2 to the host processor 7 and the driver software 11 therein, and for transferring outgoing data from the host 7 to the peripheral 2 using the host shared system memory 4. In the illustrated implementation, the descriptor system and the host shared memory 4 are used primarily for transferring outgoing (transmit) data from the driver 11 in the host 7 to the network peripheral 2 for transmission to the network 8 (Fig. 1A) and for transferring incoming (received) data from the network 8 from the network peripheral 2 to the driver 11. However, the descriptor systems and methods of the invention may be employed for transferring any type of data between a host and a peripheral.

Page 10, line 25 – page 11, line 6: Please replace the paragraph with the following:

The exemplary descriptor system comprises various queues for information exchange between the host 7 and the peripheral 2 including descriptor and status rings organized as contiguous blocks of memory locations or registers operated as circular memory rings in the ~~shared~~ host memory 4, wherein the first location in the block is considered to follow the last register. The memory locations in a descriptor or status ring need not be contiguous and other configurations are possible within the scope of the present invention. An outgoing data descriptor ring 12 in the shared memory stores outgoing data descriptors 13 indicating locations of outgoing data buffers 14, and an incoming data descriptor ring 15 stores incoming data descriptors 16 indicating locations of incoming data buffers 17 in the host shared memory 4. An incoming data status ring queue 18 stores incoming data status entries 19 corresponding to incoming data in the incoming data buffers 17.

Page 11, lines 7 – 21: Please replace the paragraph with the following:

The descriptor system also provides a control status block (CSB) 20 in the host shared memory 4. The CSB includes memory locations or registers in the host shared memory 4, which the host 7 and the driver 11 therein can access using fast memory read operations. The CSB 20 comprises an outgoing data descriptor read pointer 21, which indicates a location of an outgoing data descriptor 13 in the outgoing data descriptor ring 12. The pointer 21 and other pointers in the present invention may be a physical address of one of a particular descriptor 13 in the ring 12, or may be an offset from the address of the first location in the ring 12, or any other value indicative of the particular descriptor 13. The outgoing data descriptor read pointer 21 indicates a number of outgoing data buffers 14 to which the host processor 7 or the driver 11 therein can write outgoing data. In one implementation illustrated and described below, the pointer 21 is written by the peripheral 2 to indicate a descriptor 13 in the ring 12 just beyond the last outgoing descriptor 13 that the peripheral has processed. In this example, the host 7 can proceed to fill outgoing data buffers 14 and corresponding descriptors 13 until the location of the descriptor identified by the pointer 21 without having to directly communicate with the peripheral 2.

Page 12, lines 13 – 26: Please replace the paragraph with the following:

In the peripheral 2, a descriptor management system or unit 24 is provided, which may comprise logic circuitry and memory registers in the peripheral 2. The descriptor management system 24 comprises an outgoing data descriptor write pointer 25 and an incoming data descriptor pointer 26, implemented as memory registers in the exemplary peripheral 2. The outgoing data descriptor write pointer 25 is written by the host processor 7 and indicates a location of an outgoing data descriptor 13 in the outgoing data descriptor ring 12 and indicates a number of outgoing data buffers 14 from which the peripheral 2 can read outgoing data. The incoming data descriptor pointer 26 is also written by the host 7 and indicates a location of an incoming data descriptor 16 in the incoming data descriptor ring 15, wherein the incoming data descriptor pointer 26 indicates a number of incoming data buffers 17 to which the peripheral 2

can write incoming data. In the exemplary peripheral 2, the incoming data descriptors 16 in the incoming data descriptor ring 15 may individually indicate the location of a plurality of incoming data buffers 17 in the ~~host shared~~ memory 4 to reduce bus bandwidth usage.

Page 13, lines 5 – 16: Please replace the paragraph with the following:

The peripheral 2 reads one or more outgoing data descriptors 13 from the descriptor ring 12 according to the updated outgoing data descriptor write pointer 25, and reads outgoing data from one or more outgoing data buffers 14 in accordance therewith. The peripheral 2 then writes an updated outgoing data descriptor read pointer 21 to the CSB 20 according to the number of outgoing data buffers 14 from which the peripheral 2 has read outgoing data. The updated outgoing data descriptor write pointer 25 in the descriptor management system 24 comprises an address in the ~~host shared~~ memory 4 indicating a location in the outgoing data descriptor ring 12 just beyond the most recent outgoing data descriptor 13 written to the descriptor ring 12 by the host 7. The updated outgoing data descriptor read pointer 21 in the CSB 20 comprises an address in the memory 4 indicating a location in the outgoing data descriptor ring 12 just beyond the most recent outgoing data descriptor 13 read by the peripheral 2.

Page 14, lines 18 – 25: Please replace the paragraph with the following:

In this manner, the incoming data status pointer 22 indicates the number of incoming data buffers 17 from which the host can read incoming data, and the incoming data descriptor pointer indicates a number of incoming data buffers 17 to which the peripheral 2 can write incoming data. In the illustrated example, the exemplary CSB 20 is smaller than or equal to a cache line size for the cache memory 9. Furthermore, the exemplary peripheral 2 updates the entire cache line containing the CSB 20 in the ~~host shared~~ memory 4 in a single write operation, thereby reducing memory bandwidth usage and mitigating the number of cache line invalidations.

Page 17, lines 4 – 16: Please replace the paragraph with the following:

Further, the exemplary descriptor system may be configured to accommodate multiple quality of service (QOS) priority levels of incoming and/or outgoing data. In the exemplary system of Figs. 1A and 1B, a plurality of outgoing data descriptor rings 12, incoming data descriptor rings 15, and incoming data status rings 18 are provided in the host shared memory 4. Individual outgoing data descriptor rings 12 correspond to an outgoing data priority and store outgoing data descriptors 13 indicating locations of outgoing data buffers 14 in the host shared memory 4. With respect to incoming data, individual incoming data descriptor rings 15 correspond to incoming data priorities and store incoming data descriptors 15 indicating locations of incoming data buffers 17 and individual incoming data status rings 18 are likewise associated with a corresponding incoming data priority, where the status rings 18 store incoming data status entries 19 accordingly. The incoming data descriptors 16, moreover, may each point to a plurality of incoming data buffers 17, wherein a corresponding plurality of status entries 19 are provided in the status rings 18.

Page 18, lines 5 – 13: Please replace the paragraph with the following:

Fig. 1C provides a flow chart 30 illustrating transfer of outgoing data from the host 7 or the driver 11 therein to the peripheral 2 above using the exemplary descriptor system, and Fig. 1D illustrates transfer of incoming data from the peripheral 2 to the host 7, wherein the transfer of outgoing and incoming data between the host 7 and the peripheral 2 may occur concurrently. In Fig. 1C, outgoing data transfer operation begins at 32, where the host 7 reads a current outgoing data descriptor read pointer 21 from the CSB 20 at 34 and writes outgoing data to shared memory data buffers 14 in accordance therewith at 36. At 38, the host 7 writes the outgoing data descriptor 13 write pointer 25 in the host memory 4 peripheral 2 according to the number of outgoing buffers 14 written by the host 7. At 40, the host 7 writes updated outgoing

data descriptor write pointer 25 within the peripheral 2 according to the number of outgoing data buffers 14 written by the host 7.

Page 18, line 27 – page 19, line - 11: Please replace the paragraph with the following:

Incoming data transfer from the peripheral 2 to the host 7 is illustrated in Fig. 1D beginning at 50, with the peripheral 2 reading an incoming data descriptor 16 from the host shared memory 4 according to a current incoming data descriptor pointer 26 at 52. The peripheral 2 writes incoming data into the shared memory buffers 17 at 54 according to the incoming data descriptor 16. An individual descriptor 16 may be associated with a plurality of incoming data buffers 17, in which case the peripheral 2 need not read a separate incoming data descriptor 16 for every frame it transfers to a buffer 17. At 56, the peripheral 2 writes incoming data status entries 19 into the status ring 18 according to the current incoming data descriptor pointer 26 using full cache line writes if the computed lower limit on the available entries is greater than or equal to the computed (*e.g.*, or constant) first value, as described above. Because a single incoming data descriptor 16 may be associated with a number of incoming data buffers 17, a plurality of status entries 19 may correspond with a single incoming data descriptor 16. At 58, the peripheral 2 writes the updated incoming data status pointer 22 to the CSB 20. The peripheral 2 then writes incoming data interrupt information 23 to the CSB 20 at 60 and interrupts the host 7 at 62.

Page 19, line 23 – page 20, line - 2: Please replace the paragraph with the following:

Referring now to Figs 1E, 1F, and 2-7D, the invention is exemplified in a descriptor system provided in an exemplary network interface controller peripheral or network controller 102 having a descriptor management system ~~or unit~~ 130. The descriptor management system 130 operates in conjunction with a network driver 190 running in a host processor 112 for

transferring outgoing and incoming data (e.g., transmit and receive data) between the controller 102 and the host 112 using a shared host memory 128 as illustrated and described hereinafter. Fig. 1F provides a flow diagram illustrating an exemplary method 80 for transferring data from a peripheral to a host memory, which may be practiced in association with the exemplary network controller 102 or other peripheral devices in accordance with the present invention.

Page 20, lines 3 – 13: Please replace the paragraph with the following:

As shown in Fig. 1E, the host processor 112 may locally access the host shared memory 128 and a cache memory 115 *via* a memory/cache controller 113 and may communicate directly with the network controller 102 *via* I/O read and write operations across a system bus 106 using a host bridge 117, for example, where the exemplary system bus 106 is a system PCI-X bus. The host memory 112 includes a control status block (CSB) 196, transmit/receive data buffers 194, as well as descriptor locations for transmit descriptors 192a, receive descriptors 192b, and receive status entries 199. The CSB includes an outgoing data (transmit) descriptor read pointer TX_RD_PTR for each of four transmit (Tx) data priority (QOS) levels, a copy of the peripheral interrupt information INT0_COPY, as well as an incoming data (receive) status pointer STAT_WR_PTR[3:0] for each of four receive (Rx) data priority levels.

Page 25, line 28- page 26, line 18: Please replace the paragraph with the following:

Fig. 2 illustrates further details of the exemplary network controller 102 and Figs. 3 and 4 illustrate an exemplary single-chip implementation 102a of the network controller 102. The exemplary single-chip network controller 102a includes all the functionality and components described herein with respect to the network interface system 102. The various blocks, systems, modules, engines, etc. described herein may be implemented using any appropriate analog and/or digital circuitry, wherein one or more of the blocks, etc. described herein may be combined with other circuitry in accordance with the invention. The network controller 102 includes a 64-bit

system ~~PCI-X~~ bus interface 104 for connection with a host PCI or system ~~PCI-X~~ bus 106 that operates at a clock speed up to 133 MHz in PCI-X mode or up to 66 MHz in standard PCI mode.

The network controller 102 may be operated as a bus master or a slave. Much of the initialization can be done automatically by the network controller 102 when it reads an optional EEPROM (not shown), for example, *via* an EEPROM interface 114 (Fig. 3). The network controller 102 can be connected to an IEEE 802.3 or proprietary network 108 through an IEEE 802.3-compliant Media Independent Interface (MII) or Gigabit Media Independent Interface (GMII) 110, for interfacing the controller 102 with the network 108 *via* an external transceiver device 111. For 1000 Mb/s operation the controller 102 supports either the byte-wide IEEE 802.3 Gigabit Media Independent Interface (GMII) for 1000BASE-T PHY devices 111 or the IEEE 802.3 Ten-Bit Interface (TBI) for 1000BASE-X devices 111. The network controller 102 supports both half-duplex and full-duplex operation at 10 and 100 Mb/s rates and full-duplex operation at 1000 Mb/s.

Page 26, line 19 – page 27, line - 9: Please replace the paragraph with the following:

A host device, such as a host processor 112 on the host system ~~PCI-X~~ bus 106 in a host system 180, may interface with the network controller 102 *via* the system bus 106 and the host bridge 117. The host processor 112 includes one or more processors that can operate in a coordinated fashion. Referring also to Fig. 4, the single-chip network controller 102a may be provided on a network interface card or circuit board 182, together with a PHY transceiver 111 for interfacing the host processor 112 with the network 108 via the host bridge 117, the host bus 106, and the transceiver 111. The system ~~PCI-X~~ bus interface 104 includes PCI configuration registers used to identify the network controller 102a to other devices on the PCI bus and to configure the device. Once initialization is complete, the host processor 112 has direct access to the I/O registers of the network controller 102 for performance tuning, selecting options, collecting statistics, and starting transmissions through the host bridge 117 and the bus 106. The host processor 112 is operatively coupled with the host ~~system~~ memory 128 and a cache memory

115 via a memory/cache controller 113. One or more application software programs 184 executing in the host processor 112 may be provided with network service via layer 4 (e.g., transport layer) software, such as transmission control protocol (TCP) layer software 186, layer 3 (e.g., network layer) software 188, such as internet protocol (IP) software 188, and a software network driver 190, also running on the host processor 112. As discussed below, the network driver software 190 interacts with the host memory 128 and the network controller 102 to facilitate data transfer between the application software 184 and the network 108.

Page 27, line 20 – page 28, line - 6: Please replace the paragraph with the following:

The system ~~PCI-X~~ bus interface 104 includes a Direct Memory Access (DMA) controller 126 that automatically transfers network frame data between the network controller 102 and buffers in host ~~system~~ memory 128 via the host bus 106. The operation of the DMA controller 126 is directed by a descriptor management system ~~unit~~ 130 according to data structures called descriptors 192, which include pointers to one or more data buffers 194 in system memory 128, as well as control information. The descriptors 192 are stored in the host ~~system~~ memory 128 in queues called descriptor rings. Four transmit descriptor rings are provided for transmitting frames and four receive descriptor rings for receiving frames, corresponding to four priorities of network traffic in the illustrated controller 102. Additionally, four receive status rings are provided, one for each priority level, that facilitates synchronization between the network controller 102 and the host system. Transmit descriptors 192 control the transfer of frame data from the ~~host system~~ memory 128 to the controller 102, and receive descriptors 192 control the transfer of frame data in the other direction. In the exemplary controller 102, each transmit descriptor 192 corresponds to one network frame, whereas each receive descriptor 192 corresponds to one or more host memory buffers in which frames received from the network 108 can be stored.

Page 28, lines 15 – 30: Please replace the paragraph with the following:

Synchronization between the controller 102 and the host processor 112 is maintained by pointers stored in hardware registers 132 in the controller 102, pointers stored in a controller status block (CSB) 196 in the host ~~system~~ memory 128, and interrupts. The CSB 196 is a block of host ~~system~~ memory 128 that includes pointers into the descriptor and status rings and a copy of the contents of the controller's interrupt register. The CSB 196 is written by the network controller 102 and read by the host processor 112. Each time the software driver 190 in the host 112 writes a descriptor or set of descriptors 192 into a descriptor ring, it also writes to a descriptor write pointer register in the controller 102. Writing to this register causes the controller 102 to start the transmission process if a transmission is not already in progress. Once the controller has finished processing a transmit descriptor 192, it writes this information to the CSB 196. After receiving network frames and storing them in receive buffers 194 of the host ~~system~~ memory 128, the controller 102 writes to the receive status ring and to a write pointer, which the driver software 190 uses to determine which receive buffers 194 have been filled. Errors in received frames are reported to the host memory 128 via a status generator 134.

Page 29, lines 1 – 14: Please replace the paragraph with the following:

The IPsec module or engine 124 provides standard authentication, encryption, and decryption functions for transmitted and received frames. For authentication, the IPsec module 124 implements the HMAC-MD5-96 algorithm defined in RFC 2403 (a specification set by the Internet Engineering Task Force) and the HMAC-SHA-1-96 algorithm defined in RFC 2404. For encryption, the module implements the ESP DES-CBC (RFC 2406), the 3DES-CBC, and the AES-CBC encryption algorithms. For transmitted frames, the controller 102 applies IPsec authentication and/or encryption as specified by Security Associations (SAs) stored in a private local SA memory 140, which are accessed by IPsec system 124 via an SA memory interface 142. SAs are negotiated and set by the host processor 112. SAs include IPsec keys, which are required by the various authentication, encryption, and decryption algorithms, IPsec key

exchange processes are performed by the host processor 112. The host 112 negotiates SAs with remote stations and writes SA data to the SA memory 140. The host 112 also maintains an IPsec Security Policy Database (SPD) in the host ~~system~~ memory 128.

Page 29, line 26 – page 30, line - 2: Please replace the paragraph with the following:

A receive IPsec parser 154, associated with IPsec processor 150, performs parsing that cannot be carried out before packet decryption. Some of this information is used by a receive (Rx) checksum and pad check system 156, which computes checksums specified by headers that may have been encrypted and also checks pad bits that may have been encrypted to verify that they follow a pre-specified sequence for pad bits. These operations are carried out while the received frame is passed to the ~~system~~ ~~PCI-X~~ bus 104 via FIFO 158. The checksum and pad check results are reported to the status generator 134.

Page 30, lines 3 – 17: Please replace the paragraph with the following:

In the transmit path, an assembly RAM 160 is provided to accept frame data from the system memory 128, and to pass the data to the memory 116. The contents of a transmit frame can be spread among multiple data buffers 194 in the host memory 128, wherein retrieving a frame may involve multiple requests to the system memory 128 by the descriptor management ~~system~~ ~~unit~~ 130. These requests are not always satisfied in the same order in which they are issued. The assembly RAM 160 ensures that received chunks of data are provided to appropriate locations in the memory 116. For transmitted frames, the host 112 checks the SPD (IPsec Security Policy Database) to determine what security processing is needed, and passes this information to the controller 102 in the frame's descriptor 192 in the form of a pointer to the appropriate SA in the SA memory 140. The frame data in the host ~~system~~ memory 128 provides space in the IPsec headers and trailers for authentication data, which the controller 102 generates. Likewise, space for padding (to make the payload an integral number of blocks) is provided

when the frame is stored in the host ~~system~~ memory buffers 194, but the pad bits are written by the controller 102.

Page 30, line 18 – page 31, line - 3: Please replace the paragraph with the following:

As the data is sent out from the assembly RAM 160, it passes also into a first transmit (TX) parser 162, which reads the MAC header, the IP header (if present), the TCP or UDP header, and determines what kind of a frame it is, and looks at control bits in the associated descriptor. In addition, the data from the assembly RAM 160 is provided to a transmit checksum system 164 for computing IP header and/or TCP checksums, which values will then be inserted at the appropriate locations in the memory 116. The descriptor management system ~~unit~~ 130 sends a request to the SA memory interface 142 to fetch an SA key, which is then provided to a key FIFO 172 that feeds a pair of TX IPsec processors 174a and 174b. Frames are selectively provided to one of a pair of TX IPsec processors 174a and 174b for encryption and authentication via TX IPsec FIFOs 176a and 176b, respectively, wherein a transmit IPsec parser 170 selectively provides frame data from the memory 116 to a selected one of the processors 174. The two transmit IPsec processors 174 are provided in parallel because authentication processing cannot begin until after encryption processing is underway. By using the two processors 174, the speed is comparable to the receive side where these two processes can be carried out simultaneously.

Page 31, lines 11 – 20: Please replace the paragraph with the following:

In the single-chip implementation of Fig. 3, the controller 102a comprises a network port manager 182, which may automatically negotiate with an external physical (PHY) transceiver via management data clock (MDC) and management data I/O (MDIO) signals. The network port manager 182 ~~475~~ may also set up the MAC engine 122 to be consistent with the negotiated configuration. Circuit board interfacing for LED indicators is provided by an LED controller 171, which generates LED driver signals LED0'-LED3' for indicating various network status

information, such as active link connections, receive or transmit activity on the network, network bit rate, and network collisions. Clock control logic 173 receives a free-running 125 MHz input clock signal as a timing reference and provides various clock signals for the internal logic of the controller 102a.

Page 31, lines 21 – 29: Please replace the paragraph with the following:

A power management unit ~~175~~ 488, coupled with the descriptor management ~~system unit~~ 130 and the MAC engine 122, can be used to conserve power when the device is inactive. When an event requiring a change in power level is detected, such as a change in a link through the MAC engine 122, the power management ~~unit~~ ~~system~~ 175~~488~~ provides a signal PME' indicating that a power management event has occurred. The external serial EEPROM interface 114 implements a standard EEPROM interface, for example, the 93Cxx EEPROM interface protocol. The leads of external serial EEPROM interface 114 include an EEPROM chip select (EECS) pin, EEPROM data in and data out (EEDI and EEDO, respectively) pins, and an EEPROM serial clock (EESK) pin.

Page 34, line 27 – page 35, line - 17: Please replace the paragraph with the following:

The MAC 122 can accept and parse several header formats, including for example, IPv4 and IPv6 headers. The MAC 122 extracts certain information from the frame headers. Based on the extracted information, the MAC 122 determines which of several priority queues (not shown) to put the frame in. The MAC places some information, such as the frame length and priority information, in control words at the front of the frame and other information, such as whether checksums passed, in status words at the back of the frame. The frame passes through the MAC 122 and is stored in the memory 118 (e.g., a 32 KB RAM). In this example, the entire frame is stored in memory 118. The frame is subsequently downloaded to the system memory 128 to a location determined by the descriptor management ~~system unit~~ 130 according to the descriptors

192 in the host memory 128 (Fig. 4), wherein each receive descriptor 192 comprises a pointer to a data buffer 194 in the system memory 128. Transmit descriptors include a pointer or a list of pointers, as will be discussed in greater detail *supra*. The descriptor management ~~system unit~~ 130 uses the DMA 126 to read the receive descriptor 192 and retrieve the pointer to the buffer 194. After the frame has been written to the system memory 128, the status generator 134 creates a status word and writes the status word to another area in the system memory 128, which in the present example, is a status ring. The status generator 134 then interrupts the processor 112. The system software (*e.g.*, the network driver 190 in Fig. 4) can then check the status information, which is already in the system memory 128. The status information includes, for example, the length of the frame, what processing was done, and whether or not the various checksums passed.

Page 36, lines 1 –16: Please replace the paragraph with the following:

The frame data is read from the buffers 194 into the controller 102. To perform this read, the descriptor management ~~system unit~~ 130 reads the transmit descriptor 192 and issues a series of read requests on the host bus 106 using the DMA controller 126. However, the requested data portions may not arrive in order they were requested, wherein the ~~system~~ PCI-X interface 104 indicates to the DMU 130 the request with which the data is associated. Using such information, the assembly RAM logic 160 organizes and properly orders the data to reconstruct the frame, and may also perform some packing operations to fit the various pieces of data together and remove gaps. After assembly in the assembly RAM 160, the frame is passed to the memory 116 (*e.g.*, a 32 KB RAM in the illustrated example). As the data passes from the assembly RAM 160, the data also passes to the TX parser 162. The TX parser 162 reads the headers, for example, the MAC headers, the IP headers (if there is one), the TCP or UDP header, and determines what kind of a frame it is, and also looks at the control bits that were in the associated transmit descriptor 192. The data frame is also passed to the transmit checksum system 164 for computation of TCP and/or IP layer checksums.

Page 37, lines 8 – 20: Please replace the paragraph with the following:

In addition to the receive and transmit functions identified above, the network controller 102 may also be programmed to perform various segmentation functions during a transmit operation. For example, the TCP protocol allows a TCP frame to be as large as 64,000 bytes. The Ethernet protocol does not allow data transfers that large, but instead limits a network frame to about 1500 bytes plus some headers. Even in the instance of a jumbo frame option that allows 16,000 byte network frames, the protocol does not support a 64 KB frame size. In general, a transmit frame initially resides in one or more of the data buffers 194 in system memory 128, having a MAC header, an IP header, and a TCP header, along with up to 64 KB of data. Using the descriptor management system unit 130, the frame headers are read, and an appropriate amount of data (as permitted by the Ethernet or network protocol) is taken and transmitted. The descriptor management system unit 130 tracks the current location in the larger TCP frame and sends the data block by block, each block having its own set of headers.

Page 37, lines 21 - page 38, line - 1: Please replace the paragraph with the following:

For example, when a data transmit is to occur, the host processor 112 writes a descriptor 192 and informs the controller 102. The descriptor management system unit 130 receives a full list of pointers, which identify the data buffers 194, and determines whether TCP segmentation is warranted. The descriptor management system unit 130 then reads the header buffers and determines how much data can be read. The headers and an appropriate amount of data are read into the assembly RAM 160 and the frame is assembled and transmitted. The controller 102 then re-reads the headers and the next block or portion of the untransmitted data, modifies the headers appropriately and forms the next frame in the sequence. This process is then repeated until the

entire frame has been sent, with each transmitted portion undergoing any selected security processing in the IPsec system 124.

Page 39, line 1 – 16: Please replace the paragraph with the following:

Once the key exchange is complete, the appropriate bits reside in the SA memory 140 that indicate which key is to be used and which authentication algorithm, as well as the actual keys. In transmit mode, part of the descriptor 192 associated with a given outgoing frame includes a pointer into the SA memory 140. When the descriptor management ~~unit~~ 130 reads the descriptor 192, it sends a request to the SA memory interface 142 to fetch the key, which then sends the key to the key FIFO 172, that feeds the TX IPsec processing modules 174a and 174b, respectively. When both encryption and authentication are to be employed in transmit, the process is slightly different because the tasks are not performed in parallel. The authentication is a hash of the encrypted data, and consequently, the authentication waits until at least a portion of the encryption has been performed. Because encryption may be iterative over a series of data blocks, there may be a delay between the beginning of the encryption process and the availability of the first encrypted data. To avoid having this delay affect device performance, the exemplary network interface 102 employs two TX IPsec process engines 174a and 174b, wherein one handles the odd numbered frames and the other handles the even numbered frames in the illustrated example.

Page 41, lines 12 – 25: Please replace the paragraph with the following:

The network interface 102 avoids many read latencies by replacing read operations with write operations. Write operations are not as problematic because they can take place without involving the processor 112. Thus when write information is sent to a FIFO, as long as the writes are in small bursts, the network controller 102 can take the necessary time to execute the writes without negatively loading the processor. To avoid read operations during a transmit operation, the driver creates a descriptor 192 in the system memory 128 and then writes a pointer

to that descriptor to the register 132 of the network controller 102. The DMU 130 of the controller 102 sees the contents in the register 132 and reads the necessary data directly from the system memory 128 without further intervention of the processor 112. For receive operations, the driver software 190 identifies empty buffers 194 in the system memory 128, and writes a corresponding entry to the register 132. The descriptor management system unit 130 writes to pointers in the transmit descriptor rings to indicate which transmit descriptors 192 have been processed and to pointers in the status rings to indicate which receive buffers 194 have been used.

Page 42, line 24 – page 43, line 7: Please replace the paragraph with the following:

As shown in Fig. 5A, the descriptors 192 individually include pointers to one or more data buffers 194 in the system memory 128, as well as control information, as illustrated in Figs. 5E-5G. Synchronization between the controller 102 and the software driver 190 is provided by pointers stored in the controller registers 132, pointers stored in the CSB 196 in the system memory 128, and interrupts. In operation, the descriptor management system unit 130 in the controller 102 reads the descriptors 192 *via* the DMA controller 126 of the bus interface 104 in order to determine the memory location of the outgoing frames to be transmitted (*e.g.*, in the data buffers 194) and where to store incoming frames received from the network 108. The CSB 196 is written by the network controller 102 and read by the driver 190 in the host processor 112, and the descriptor management registers 132 are written by the driver 190 and read by the descriptor management system unit 130 in the controller 102. The exemplary descriptor system generally facilitates information exchange regarding transmit and receive operations between the software driver 190 and the controller 102.

Page 43, line 19– page 44, line 6: Please replace the paragraph with the following:

Fig. 5C illustrates registers 132 related to the descriptor management system unit 130 in the controller 102. Transmit descriptor base pointers TX_RING[3:0]_BASE include the memory addresses of the start of the transmit descriptor rings of corresponding priority, and the lengths of the transmit descriptor rings are provided in TX_RING[3:0]_LEN registers. Transmit descriptor write pointers are stored in registers TX_WR_PTR[3:0], where the driver software 190 updates these registers to point just beyond the last QWORD that the driver has written to the corresponding transmit descriptor ring. Receive descriptor base pointers RX_RING[3:0]_BASE include the memory address (e.g., in host memory 128) of the start of the receive descriptor rings of corresponding priority, and the lengths of these receive descriptor rings are provided in RX_RING[3:0]_LEN registers. Receive descriptor write pointers RX_WR_PTR[3:0] are updated by the driver 190 to point just beyond the last QWORD that the driver has written to the corresponding receive descriptor ring. Receive status ring base pointer registers STAT_RING[3:0]_BASE indicate the memory address of the receive status rings, and STAT_RING[3:0]_BASE indicate the lengths of the corresponding receive status rings 199 in memory 128. RX_BUF_LEN indicates the number of QWORDS of the receive data buffers 194, where all the receive data buffers 194 are of the same length, and CSB_ADDR indicates the address of the CSB 196 in the host memory 128.

Page 44, lines 7– 26: Please replace the paragraph with the following:

To further illustrate descriptor management operation in data transmission, Fig. 5D illustrates the host memory 128 and the descriptor management system unit 130, including an exemplary transmit descriptor ring in the host memory 128 and the corresponding descriptor registers 132 in the descriptor management system unit 130 of the controller 102. In addition, Figs. 5E and 5F illustrate an exemplary transmit descriptor 192a and control flags thereof, respectively. In the transmit descriptor 102 of Fig. 5E, BUF1_ADR[39:0] includes an address in the host memory 128 of the first data buffer 194 associated with the descriptor 192a. The descriptor 192a also includes transmit flags (TFLAGS1, Figs. 5E and 5F) 193, which include a MORE_CTRL bit to indicate inclusion of a second 64-bit control word with information relating

to virtual local area network (VLAN) operation and TCP segmentation operation. An ADD_FCS/IVLEN1 bit and an IVLEN0 bit are used for controlling FCS generation in the absence of IPsec processing, or to indicate the length of an encapsulation security protocol (ESP) initialization vector (IV) when IPsec security and layer 4 processing are selected. An IPCK bit is used to indicate whether the controller 102 generates a layer 3 (IP layer) checksum for transmitted frames, and an L4CK flag bit indicates whether the controller 102 generates a layer 4 (e.g., TCP, UDP, etc.) checksum. Three buffer count bits BUF_CNT indicate the number of data buffers 194 associated with the descriptor 192a, if less than 8. If more than 8 data buffers 194 are associated with the descriptor 192a, the buffer count is provided in the BUF_CNT[7:0] field of the descriptor 192a.

Page 45, lines 15 – 28: Please replace the paragraph with the following:

When the network software driver 190 writes a descriptor 192 to a descriptor ring, it also writes to a descriptor write pointer register 132 in the descriptor management system unit registers 132 to inform the controller 102 that new descriptors 192 are available. The value that the driver writes to a given descriptor management register 132 is a pointer to a 64-bit word (QWORD) in the host memory 128 just past the descriptor 192 that it has just written, wherein the pointer is an offset from the beginning of the descriptor ring measured in QWORDS. The controller 102 does not read from this offset or from anything beyond this offset. When a transmit descriptor write pointer register (e.g., DMU register 132, such as TX_WR_PTR1 in Fig. 5D) has been written, the controller 102 starts a transmission process if a transmission is not already in progress. When the transmission process begins, it continues until no unprocessed transmit descriptors 192 remain in the transmit descriptor rings. When the controller 102 finishes a given transmit descriptor 192, the controller 102 writes a descriptor read pointer (e.g., pointer TX_RD_PTR1 in Fig. 5D) to the CSB 196.

Page 46, line 12 – page 47, line -2: Please replace the paragraph with the following:

Referring also to Fig. 5G, an exemplary receive descriptor 192b is illustrated, comprising a pointer BUF_ADR[39:0] to a block of receive buffers 194 in the host ~~system~~ memory 128, and a count field BUF_MULT[7:0] indicating the number of buffers 194 in the block, wherein all the receive buffers 194 are the same length and only one buffer is used for each received frame in the illustrated example. If the received frame is too big to fit in the buffer 104, the frame is truncated, and a TRUNC bit is set in the corresponding receive status ring entry 199. Fig. 5H illustrates an exemplary receive descriptor ring comprising an integer number n receive descriptors 192b for storing addresses pointing to n receive data buffers 194 in the host memory 128. The registers 132 in the descriptor management ~~system~~ ~~unit~~ 130 of the controller 102 include ring base and length registers (RX_RING1_BASE and RX_RING1_LEN) corresponding to the receive descriptor ring, as well as a receive write pointer register (RX_WR_PTR1) including an address of the next unused receive descriptor 192b in the illustrated descriptor ring, and a receive buffer length register (RX_BUF_LEN) including the length of all the buffers 194. The descriptor management ~~system~~ ~~unit~~ 130 also has registers 132 (STAT_RING1_BASE and STAT_RING1_LEN) related to the location of the receive status ring having entries 199 corresponding to received data within one or more of the buffers 194. The control status block 196 in the host memory 128 also includes a register STAT_WR_PTR1 whose contents provide the address in the receive status ring of the next unused status ring location, wherein the receive status ring is considered empty if STAT_WR_PTR1 equals RX_WR_PTR1.